ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/225329391

Parker–Sochacki method for solving systems of ordinary differential equations using graphics processors

Article *in* Numerical Analysis and Applications · July 2011

DOI: 10.1134/S1995423911030049

CITATIONS
13

READS
1,420

2 authors, including:

Evgeni Alekseevich Nurminski
Far Eastern Federal University
123 PUBLICATIONS **745** CITATIONS

SEE PROFILE

All content following this page was uploaded by Evgeni Alekseevich Nurminski on 18 November 2014.

The user has requested enhancement of the downloaded file.

ISSN 1995-4239, Numerical Analysis and Applications, 2011, Vol. 4, No. 3, pp. 223–233. © Pleiades Publishing, Ltd., 2011.
Original Russian Text © E.A. Nurminskii, A.A. Buryi, 2011, published in Sibirskii Zhurnal Vychislitel'noi Matematiki, 2011, Vol. 14, No. 3, pp. 277–290.

Parker–Sochacki Method for Solving Systems of Ordinary Differential Equations Using Graphics Processors

E. A. Nurminskii* **and A. A. Buryi****

Institute of Automatics and Control Processes, FEB RAS, ul. Radio 5, Vladivostok, 690041 Russia

Received October 18, 2010; in final form, January 12, 2011

Abstract—The Parker–Sochacki method, which is used for solving systems of ordinary differential equations, and implementation of this method on graphics processors are described. The solution to the classical N-body problem is considered as a test. The algorithm makes it possible to effectively use massive parallel graphics processors and provides acceptable accuracy with multiple time reduction, as compared to processors of a conventional architecture.

DOI: 10.1134/S1995423911030049

Keywords: *numerical integration of ODE systems, parallel computing.*

1. PARKER–SOCHACKI METHOD

The Parker–Sochacki method for solving systems of ordinary differential equations (ODEs) numerically allows obtaining the Maclaurin series for solving ODE systems with a polynomial right-hand side. Recent revival of interest to this algorithm can be noted [1, 2]. In some cases, the use of the Maclaurin series is more preferable than the use of conventional difference schemes. Another advantage of the method is the possibility of its parallelization.

We consider the classical problem of numerical integration of an ODE system

$$\frac{du}{dt} = f(u,t) \tag{1}$$

on the time interval $t \in [0,T]$ with a specified initial condition $u(0) = u^0 \in E^n$ and the right-hand side of the differential equation $f : E^n \times [0,T] \to E^n$, where E^n is the Euclidean n-dimensional space. Standard conditions of existence and uniqueness of the solution to problem (1) are assumed to be valid: $f(u,t)$ is a continuous function with respect to the set of its arguments and a Lipschitz function with respect to u, i.e., $\|f(t,u') - f(t,u'')\| \le L\|u' - u''\|$ for a certain L and all $t \in [0,T]$. If these conditions are satisfied, then not only a unique solution to problem (1) with the specified initial condition exists, but also this solution can be obtained rather rapidly by applying a converging Picard process (see, e.g., [4]):

$$u^{k+1}(t) = u^0 + \int_0^t f(s,u^k(s))\,ds, \quad k = 0,1,\ldots, \tag{2}$$

which can be started from an arbitrary approximation $u^0(t)$, $t \in [0,T]$.

Although this method was used for the theoretical investigation of ODEs, for instance, with the use of the theory of compressing images, it has not been applied in practice because of the exponential growth of computational complexity of iterations (2). The situation changed appreciably when Parker and Sochacki [3] noted that, in the case of numerical integration of ODEs with a polynomial right-hand side f, each $u^k(t)$, $k = 0,1,\ldots$, is a polynomial, and it is sufficient to use a finite number of Picard iterations, which depends on the power index of this polynomial, to obtain a specified number

*E-mail: nurmi@dvo.ru
**E-mail: buryalex@dvo.ru

of exact values of the Maclaurin expansion coefficients to solve Eq. (1). With an admissible estimate of the Maclaurin series error, it is possible to calculate the system coordinates at an arbitrary time without using interpolation schemes, etc.

The following notions and designations are used in this paper.

Definition 1. A polynomial of n variables is a finite formal sum of the form

$$p(x) = \sum c_I x_1^{i_1} x_2^{i_2} \dots x_n^{i_n}, \tag{3}$$

where $I = (i_1, i_2, \dots, i_n) \in \mathbb{Z}_+^n$, is called a *multi-index*, $c_I \in \mathbb{R}$ is the polynomial coefficient, $\mathbb{Z}_+$ is the set of nonnegative integer numbers, and $\mathbb{R}$ is the set of real numbers.

In what follows, we deal with ODE (1), where a polynomial function $P : E^n \to E^n$ of the vector argument from E^n is used as the right-hand side. For such functions, $\Pi_k P(x)$ means the kth component of $P(x)$. For the vector v, the kth component is designated by v_k.

Definition 2 [3]. $P : E^n \to E^n$ is called a *polynomial function* if there exists

$$T : \{k \mid k \in \mathbb{N}, k \le n\} \to 2^{\Phi},$$

where $\mathbb{N}$ is the set of natural numbers,

$$\Phi = \{f : \{k \mid k \in \mathbb{N}, k \le n\} \to \mathbb{N} \cup \{0\}\},$$

$$c : \{(k, q) \mid k \in \mathbb{N}, k \le n, q \in T(k)\} \to \mathbb{R} \setminus \{0\},$$

and if there exists $v \in E^n$ such that, if $k \le n$ and $x \in E^n$, then

$$\Pi_k P(x) = v_k + \sum_{q \in T(k)} c(k, q) \prod_{i=1}^{n} x_i^{q(i)}.$$

For a given initial value u^0, using the following recurrent relations, we define

$$P_s(t) = \begin{cases} u^0 & \text{for } s = 1, \\ u^0 + \int\limits_0^t P(P_{s-1}(\tau))\, d\tau & \text{for } s > 1, \end{cases} \tag{4}$$

the sequence $P_s : E^n \to E^n$, $s = 1, 2, \dots$. The power indices j_m and j_r of the polynomial functions P_m and P_r depend on the number of the Picard iterations (4) and on the power of P. These indicators do not decrease with increasing iterations. The following theorem is valid.

Theorem 1 [3]. *Let us assume that $P : E^n \to E^n$ is a polynomial function. Let $u^0 \in E^n$. If $r \in \mathbb{N}$, $m > r$, and $k \le n, j_r \le j_m$, then*

$$(\Pi_k P_r)(t) = u_k + \sum_{i=1}^{j_r} a_{k,r,i} t^i, \qquad (\Pi_k P_m)(t) = u_k + \sum_{i=1}^{j_m} a_{k,m,i} t^i.$$

Further, if $l < r$, then

$$a_{k,r,l} = a_{k,m,l}, \tag{5}$$

where $a_{k,r,i} \in E$ and $a_{k,m,i} \in E$.

Statement (5) of the theorem means that further Picard iterations (after the rth iteration) do not change the previous $a_{k,r,i}$ coefficients in the expansion. As was demonstrated in [3], Theorem 1 guarantees, for ODEs with the right-hand side in the form of a polynomial function and initial conditions specified at the zero point, the sth iteration of the resultant polynomial coincides with s first terms of the Maclaurin series. The next theorem extends the described result to the cases with initial conditions specified at a point other than zero.

Theorem 2 [3]. *Let P be a polynomial function on E^{n+1} and $y : E^n \to \mathbb{R}$ such that, if $t > T$, $t \in D(y')$, then*

$$y'(t) = P(t, y(t)), \quad y(T) = x.$$

Let g be a function on E^{n+1} such that, if $t \geq 0$, $t \in D(g')$, then

$$g'(t) = (1, P(g(t))), \quad g(0) = (T, x).$$

Let us use the following definition:

$$P_s(t) = \begin{cases} u_0 & \text{for } s = 1, \\ u_0 + \int\limits_t^T P \circ (I, p_s)\, d\tau\} & \text{for } s > 1, \end{cases} \tag{6}$$

$$P_s : E \to E,$$

$$Q_s(t) = \begin{cases} g(0) & \text{for } s = 1, \\ g(0) + \int\limits_0^t (1, P \circ (q_s))\, d\tau\} & \text{for } s > 1, \end{cases} \tag{7}$$

$$Q_s : E \to E.$$

Therefore, if $s \in \mathbb{N}$, then

$$\Pi_{2Q_s}(t - T) = P_s(t).$$

According to [3], Theorem 2 allows the solution to be obtained in the form of Taylor series for the initial condition at the point $t - T$. The next definition is important for analytical functions (functions that coincide with their Taylor series in the neighborhood of an arbitrary point of the domain of definition), which are not polynomial.

Definition 3 [3]. Let us assume that $y : [0, T) \to E$ is an analytical function. The function y is called projectively polynomial (in terms of [3]) if there exists a polynomial $P : E^n \to E^n$, $n \in \mathbb{N}$, and a function $w : E^n \to E^n$ such that $w' = \frac{dw}{dt} = P(w)$, for which there exists $k \leq n$ such that $y = \Pi_k w$.

One can combine projectively polynomial functions, still remaining in the same class.

Theorem 3 [3]. *Let us assume that f and g are projectively polynomial functions. Then, $f + g$, fg, and $f(g)$ are projectively polynomial functions.*

There are several conclusions from Theorems 1–3, which are important for applications.

1. The solution in the form of a polynomial is unique and, therefore, coincides with the Maclaurin series.
2. The properties of a projectively polynomial function are retained for the sum, product, and derivative of the projectively polynomial function.
3. To calculate the $(n + 1)$th term of the approximating polynomial in the Parker–Sochacki method, it is necessary to calculate only n previous terms.
4. For solutions in the form of analytical functions, the solution can be obtained with arbitrary accuracy on a finite interval.
5. The solution obtained by the Picard iterations satisfies the local Lipschitz condition and guarantees that the resultant expansion is a Maclaurin series.

As Parker and Sochacki noted in [3], by the moment of writing the paper, they had not encountered any ODE system that could not be brought to a polynomial form. They did not give any estimates, however, how large the class of projectively polynomial functions is.

2. EXAMPLE OF USING THE PARKER–SOCHACKI ALGORITHM

As an example of using the Parker–Sochacki algorithm, let us consider the ODE

$$\frac{dx}{dt} = \frac{1}{1+t} \tag{8}$$

with the initial condition $x(0) = 0$. The solution of this equation is $x(t) = \ln(t+1)$. To apply the Parker–Sochacki method, we transform the system to a polynomial form

$$\frac{dx}{dt} = u, \qquad \frac{du}{dt} = -u^2$$

with the initial conditions $x(0) = 1$ and $u(0) = 1$. We write the solution $x(t)$, $u(t)$ in the form of the Maclaurin series

$$x(t) = \sum_{k=0}^{n} x_k t^k, \quad u(t) = \sum_{k=0}^{n} u_k t^k.$$

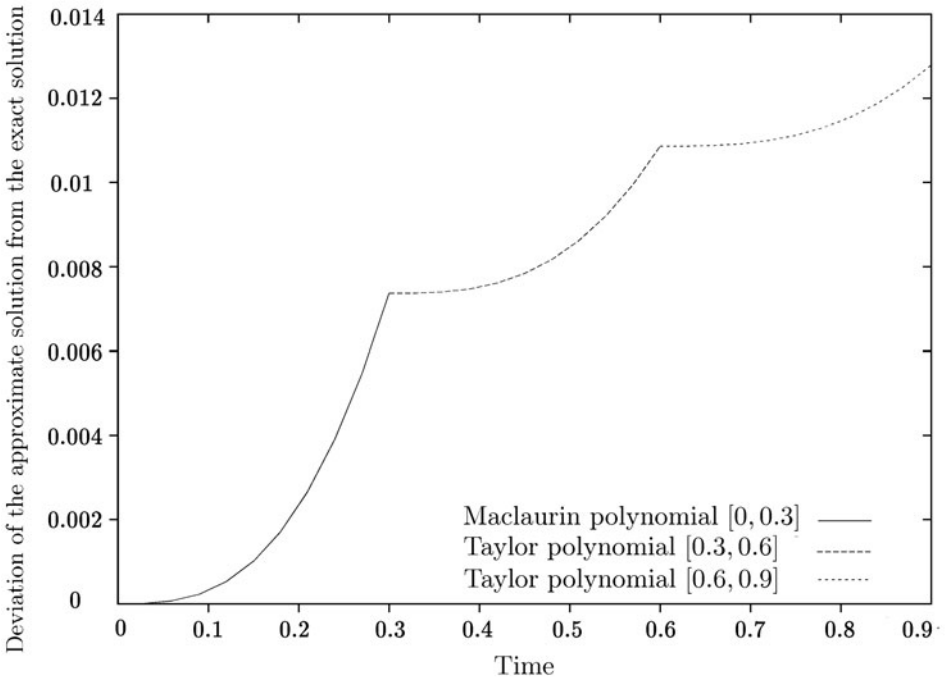

Fig. 1. Deviation of the approximate solution from the exact solution for cubic Maclaurin–Taylor polynomials. The time step is 0.3.

Table 1. Coefficients of the Maclaurin–Taylor polynomials

Coefficients	Interval	$k=0$	$k=1$	$k=2$	$k=3$
x_k	$[0.0, 0.3]$	1	$-1/2$	$1/3$	$-1/4$
u_k		-1	1	-1	1
x_k	$[0.3, 0.6]$	0.262	0.769	-0.295	0.151
u_k		-0.591	0.455	-0.350	0.269
x_k	$[0.6, 0.9]$	0.470	0.625	-0.195	0.081
u_k		-0.390	0.244	-0.152	0.095

It follows from (4) that the coefficients x_k and u_k can be calculated by the formulas

$$k = 1, \ldots, n; \quad x_k = \frac{u_{k-1}}{k}; \quad u_k = -\frac{1}{k}\sum_{l=0}^{k-1} u_l u_{k-l-1}; \quad x_0 = 0, \quad u_0 = 1. \tag{9}$$

In what follows, the segment of the Maclaurin series limited by $(m+1)$ terms is called a Maclaurin polynomial of the mth order; for the Taylor series, we call it a Taylor polynomial. The interval where the Maclaurin or Taylor polynomial is calculated is called a time step of integration. The entire time period when integration is performed can be divided into several time steps. An important issue is the relationship between the power of the Maclaurin polynomial and the length of the time step, which determines the accuracy of the solution obtained. The extreme strategies are either choosing a high power of the polynomial with a large time step or dividing the necessary interval of integration into smaller time steps and using polynomials of a comparatively power at each time step. It is also possible to vary both the polynomial degree and the size of the time interval simultaneously, which was done, e.g., in [5]

For a qualitative study of this issue, we performed numerical experiments with solving ODE (8) on the interval $[0, 0.9]$ with a time step 0.3. Approximation of the solution of (8) by a cubic Maclaurin–Taylor polynomial was considered at each time step. The polynomial coefficients are listed in Table 1. Figure 1 shows the deviation of the approximate solution from the exact solution.

The solution is substantially different from the exact solution, especially at the right end of the interval. An additional experiment shows that the solution accuracy for this equation can be increased by two methods: either increasing the polynomial power or decreasing the time step. As the indicator of the solution accuracy, we use the integral norm L_2: $\delta(f, f^*) = \|f - f^*\|_2 = \int\limits_a^b (f(x) - f^*(x))^2 dx$, where $f(x)$ and $f^*(x)$ are the exact and approximate solutions on the interval $[a, b]$. To calculate the integral in the error formula, we used the Simpson's method, which provided good agreement with the control analytical calculation. Two variants of the numerical solution were considered. In the first case, the time step was fixed, and the power of the approximating polynomials was changed. The error was calculated individually for each time step. Figure 2 shows the behavior of the error $\delta(f, f^*)$ as the Taylor polynomial

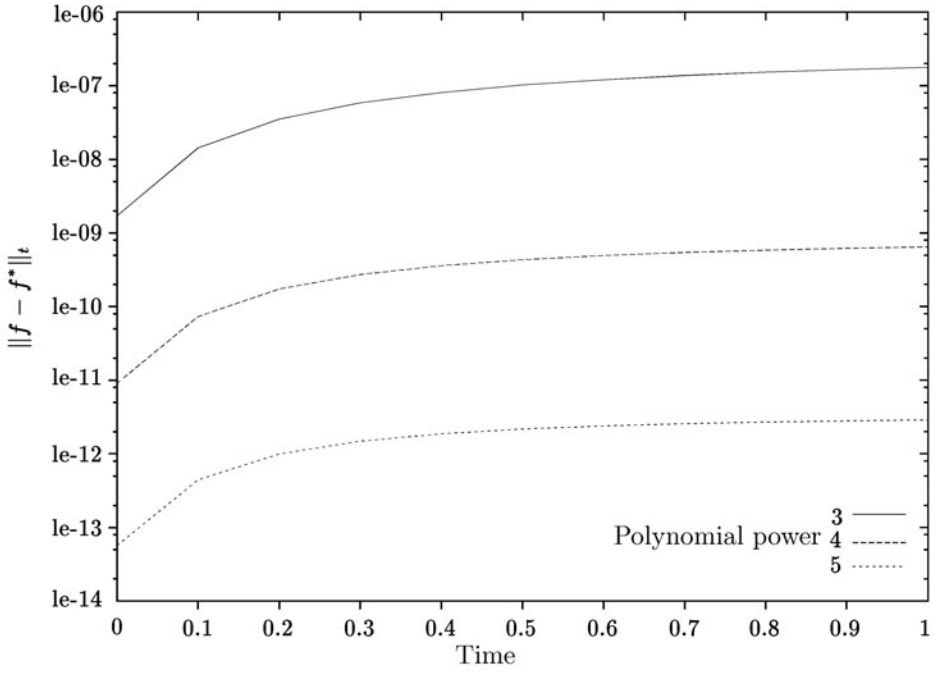

Fig. 2. Dependence of the integral error $\delta(f, f^*)$ on time for different powers of the Maclaurin–Taylor polynomials. The time step is 0.1.

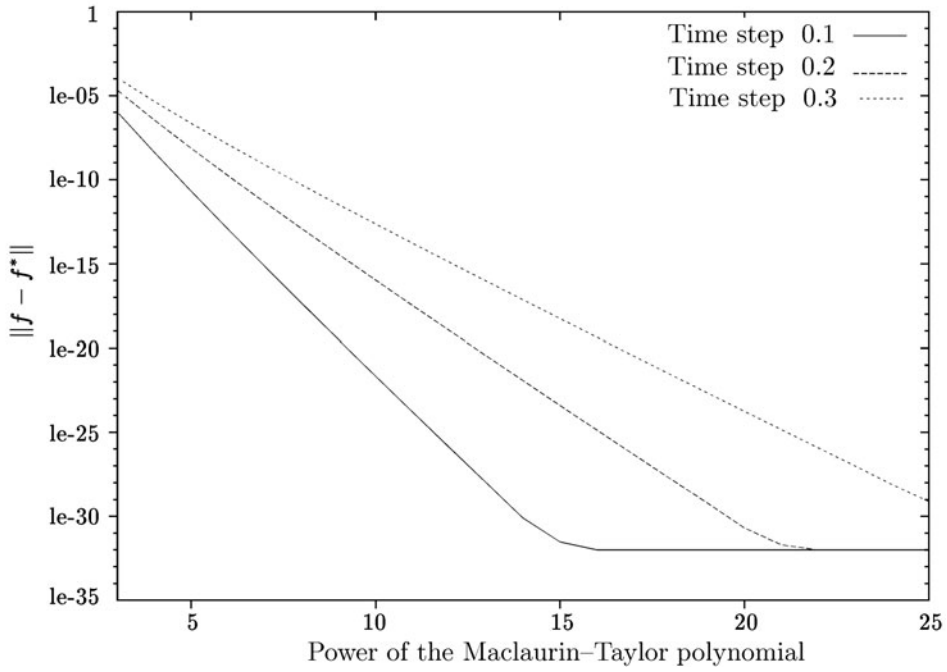

Fig. 3. Dependence of the integral error $\delta(f, f^*)$ on time for different powers of the Maclaurin–Taylor polynomials.

power is increased from 3 to 5. Even a small power of the approximating polynomial ensures a rather small error. It is seen that the error remains small inside the time step interval as well. In the second case, the polynomial power was fixed, and the time step was changed. In this case, the error was calculated for the entire interval on which the solution was sought. Figure 3 shows the behavior of the error $\delta(f, f^*)$ with decreasing the time step. Beginning from a certain moment, depending on the time step, an increase in the polynomial power does not reduce the error, which reaches the level of computer accuracy. The error logarithm decreases linearly, which agrees with the order of the residual term in the Taylor series for $\ln(x+1)$.

2.1. Application of the Parker–Sochacki Method in the N-Body Problem

As a more realistic example of using the Parker–Sochacki method, we consider the classical gravitational N-body problem. This problem is a good example for illustrating both the application of the Parker–Sochacki method and the capabilities of graphics processors. Other methods of solving this problem with the use of graphics processors were described in [6–9].

N material points interacting in accordance with Newton's gravitation law in the absence of other forces are considered in the classical formulation. The velocities and coordinates of the point at the time $t = 0$ are specified as the initial data. This system can be described by the equations

$$\frac{dx_{ij}}{dt} = v_{ij}, \qquad \frac{dv_{ij}}{dt} = \sum_{k \neq j}^{N} GM_k \frac{x_{ik} - x_{ij}}{\left(\sum\limits_{i=1}^{3} (x_{ik} - x_{ij})^2 \right)^{3/2}},$$

where N is the number of points, $i = 1, 2, 3$, $j = 1, 2, 3, \ldots, N$, is the index of the point, M_k is the mass of the kth point, G is the gravitation constant, x_{ij} is the ith Cartesian coordinate of the jth point, and v_{ij} is the ith coordinate of velocity of the jth point.

Rudmin [10] proposed solving the above-described equations by the Parker–Sochacki method for the solar system, which can be done by introducing new variables u_{ij}, $i = 1, 2, 3$, $j = 1, 2, \ldots, N$,

$$\frac{dx_{ij}}{dt} = v_{ij}, \quad \frac{dv_{ij}}{dt} = \sum_{k=1}^{N} m_k (x_{ik} - x_{ij}) u_{jk}^3, \quad \frac{du_{jk}}{dt} = -u_{jk}^3 \sum_{i=1}^{3} (x_{ik} - x_{ij})(v_{ik} - v_{ij}). \tag{10}$$

System (10) contains a total of $N(N-1)/2+6N$ equations. Representing the coordinate variables in the form of the polynomials

$$x(t)_{ij}=\sum_{l=0}^{M}\alpha_{ijl}t^l, \tag{11}$$

we obtain a polynomial presentation for velocities

$$v(t)_{ij}=\sum_{l=0}^{M}\beta_{ijl}t^l, \tag{12}$$

where

$$\alpha_{ijl}=\frac{\beta_{i,j,l-1}}{l}, \tag{13}$$

$(i,j,k,l)\in[1,3]\times[1,N]^2\times[1,M]$,

$$\begin{aligned}\beta_{ijl} &= \sum_{k=1}^{N_p}\frac{m_k}{l}\sum_{q=0}^{l-1}(\alpha_{ikq}-\alpha_{ijq})(u_{j,q,l-q-1})^3,\\ u^2_{jkl} &= \sum_{q=0}^{l}u_{jkq}u_{j,k,l-q}, \qquad u^3_{jkl}=\sum_{q=0}^{l}u^2_{jkq}u_{j,k,l-q},\end{aligned} \tag{14}$$

$$u_{jkl} = -\frac{1}{l}\sum_{q=1}^{l-1}u^3_{jkq}A_{j,k,l-q}, \tag{15}$$

where

$$A_{jkl}=\sum_{q=0}^{l}\sum_{i=1}^{3}(\alpha_{ijq}-\alpha_{ikq})(\beta_{i,j,l-q}-\beta_{i,k,l-q}).$$

Equations (13)–(15) allow the N-body problem to be solved by the Parker–Sochacki method.

3. PARALLEL IMPLEMENTATION OF THE PARKER–SOCHACKI METHOD ON GRAPHICS PROCESSORS

For implementation of the parallel version of the Parker–Sochacki algorithm, we used graphics processor units (GPUs), which are now rather popular as a high-performance computational platform. As a particular graphics processor, we used video cards of one of the leading producers in this field, NVIDIA, which can also be used for nongraphical computations. For this purpose, we used a set of libraries and programs of the Compute Unified Device Architecture (CUDA) [11, 12]), which provide access to the GPU memory and computational resources. The basic programming language in this system is a simplified version of the C language.

The GPU is a processor designed on the basis of the Single Instruction-Multiple Data (SIMD) architecture, i.e., it is a vector processor. For this reason, it has better performance than the conventional GPUs on a certain class of problems. The GPU can simultaneously perform a large number of processes, each performing the same command, but with different data.

The technology of writing codes consists in writing a C code executed on a host computer, i.e., a computer to which the accelerator including the NVIDIA processor is connected, and the so-called kernel procedures executed on the GPU proper.

Let us consider the NVIDIA GPU/CUDA system in more detail. The following terms are used in programming with the use of the CUDA. A thread is understood as a sequence of commands and data to be processed. A warp is a set of 32 threads executed in parallel on a multiprocessor. A block is a set of 64 to 512 threads. A grid is a set of blocks. Such a hierarchy is related to the specific features of GPU hardware. As the number of blocks that can be simultaneously processed by the GPU differs,

depending on the model, such arrangement allows us to avoid using a particular version of the video card. Each video card consists of several texture processors. Each texture processor consists of several thread multiprocessors. Each multiprocessor has a moderate amount of memory, which is called a shared memory. The programmer can control the behavior of this memory. Therefore, this memory is not an analog of the cache in usual processors. This memory is used for exchange of information between the threads in one block. There is also a common memory, which has large delays in access time and lower speed. Also there is a texture cache, which can be used for storage of constants. The computational complexity of the Parker–Sochacki algorithm is

$$O(N^2K^2), \tag{16}$$

where N is the number of bodies and K is the power of the Maclaurin polynomials. The memory requirement is $O(N^2K)$. Technically, the memory requirements are more important for implementation on the GPU than the algorithm complexity. At the moment, video cards and specialized cards with a memory volume from 1 to 6 GB are available, which allows one to solve the N-body problem with the dimension from 10,000 to 60,000 objects and with the polynomial power equal to 3. The linear dependence of memory on the polynomial power allows polynomials to power 18 to be used for a system of 10,000 objects. Solving such a problem requires 6 GB of memory. There are some possibilities to alleviate memory requirements. The first one is canceling the division of u^3 into two parts. If this division is used, it is necessary to store two additional arrays. It is also possible to cancel the storage array A, which has the same dimension as u^2 and u^3. Thus, it is possible to use only one array for storage of the values of u. As a result, u is calculated in the following manner:

$$u_{jkm} = -\sum_{l=0}^{m}\left(\sum_{p=0}^{l}\left(\sum_{q=0}^{p-1}\frac{u_{jkq}^3 A_{j,k,p-1-q}}{p}\sum_{r=0}^{l-p-1}\frac{u_{jkr}^3 A_{j,k,l-p-1-r}}{l-p}\right)\sum_{t=0}^{m-l-1}\frac{u_{jkt}^3 A_{j,k,m-l-1-t}}{m-l}\right).$$

The next step in reducing memory requirements is the use of the property of symmetry of u. Usual vectors and recalculation of indices were used for storage of matrices and three-dimensional arrays in program implementation. Thus, to use the property of symmetry, it was sufficient to change the formulas for recalculating the element indices.

4. COMPUTATIONAL EXPERIMENTS

The computational experiments were performed with the use of NVIDIA 280 GTX (clock frequency 1296 MHz, 240 thread multiprocessors, and 1 GB shared memory). To compare the performance with CPU, we used Intel(R) Xeon(R) CPU E7330 with a frequency of 2.40 GHz.

The data generator formed a system of a massive body with bodies of much smaller mass rotating around the large body. The location, velocity, and mass of the bodies were specified in a random manner. In the experiments performed, the systems of bodies had the following parameters: the number of bodies was varied from 100 to 500 with a step of 100; the power of the Maclaurin polynomials was changed from 3 to 25. The number of equations reached 127,750 for a system of 500 bodies. The times of program execution on the GPU and CPU versions of the program were measured in each experiment. According to [10], the unit of the model time is the interval equal to 58 astronomical days. As the main goal of the study was to compare the times of algorithm execution on the GPU and CPU, the following time parameters were used: the time step corresponded to one hundred thousandth of the model time; a total of 10,000 time steps were used, which was equal to 5.8 astronomical days.

The overall situation with comparisons of the CPU and GPU versions of implementation of the Parker–Sochacki method is illustrated in Figs. 4–7. It is seen in Fig. 4 that the time of program execution does not increase with increasing polynomial power in the cases with 100 and 200 bodies. This fact can be explained as follows: for these systems, the number of bodies is smaller than the number of thread multiprocessors, and there is a considerable reserve of GPU resources. Figure 6 shows the CPU/GPU performance ratio. This figure shows that the advantage of the GPU over the CPU increases with increasing system dimension. As a whole, it is seen in Fig. 7 that the efficiency of program execution on the GPU increases linearly, beginning from 300 bodies. The experiments performed show that the GPU version of implementation of the Parker–Sochacki algorithm for the N-body problem offers a significant advantage over the CPU version in terms of the execution time.

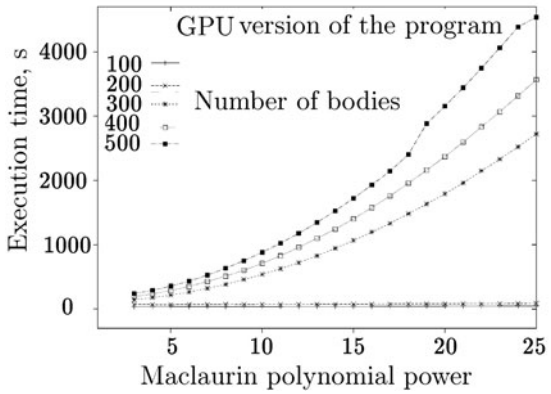

Fig. 4. Execution time of the GPU version in seconds versus the Maclaurin polynomial power.

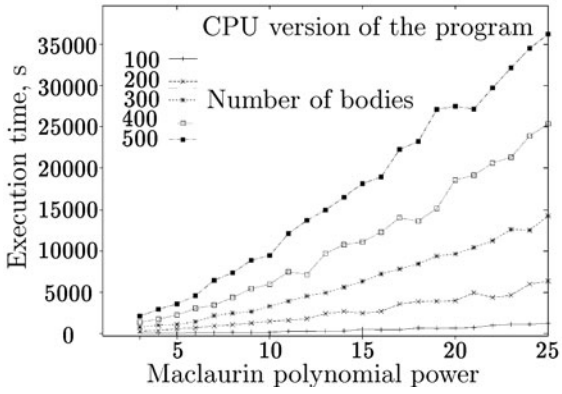

Fig. 5. Execution time of the CPU version in seconds versus the Maclaurin polynomial power.

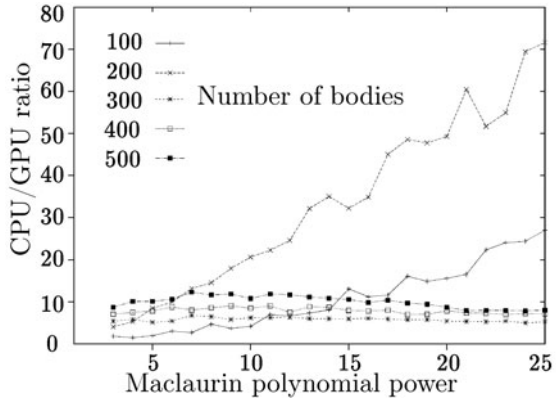

Fig. 6. CPU/GPU performance ratio.

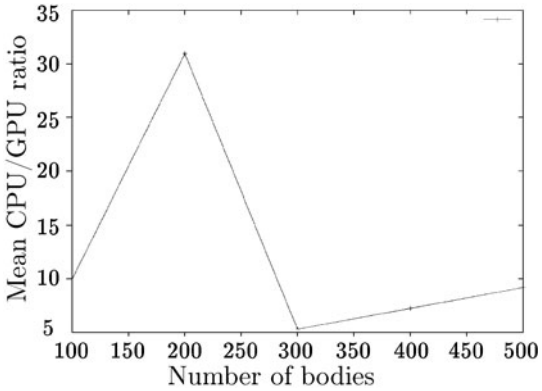

Fig. 7. Mean CPU/GPU performance ratio.

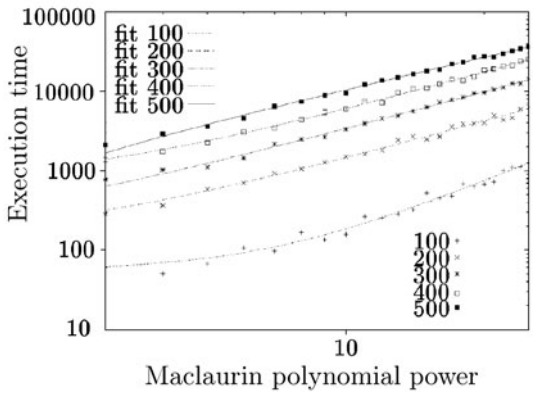

Fig. 8. Execution time of the CPU version versus the Maclaurin polynomial power on a logarithmic scale.

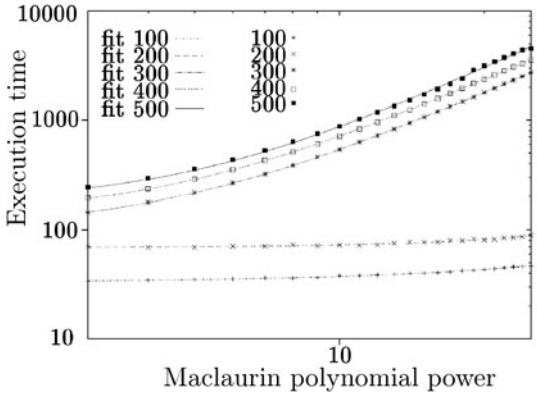

Fig. 9. Execution time of the GPU version versus the Maclaurin polynomial power on a logarithmic scale.

A more detailed study of the results of the numerical experiments (see Figs. 8 and 9) shows that the dependence of the program execution time both for the CPU and GPU is fairly well described by a power dependence on the Maclaurin polynomial power $t(k) = a + bk^{\alpha}$ with the value of the power index α of the order of 2, depending on the number of bodies N, which is in overall agreement with the theoretical estimate of the algorithm complexity (16). The values of α obtained by the least squares method for different numbers of bodies are listed in Table 2.

The decrease in α_{CPU} from 2.4 to 1.3 with the number of bodies increased from 100 to 500 can be

Table 2. Values of α for the CPU and GPU program versions

N	100	200	300	400	500
α_{CPU}	2.4	1.6	1.5	1.6	1.3
α_{GPU}	1.2	1.7	1.9	1.9	2.0

explained by two factors. The first one is the fact that implementation of the Parker–Sochacki algorithm is a linear code, which is well supported by hardware of the majority of modern CPUs. The second factor is the presence of a long pipeline in the architecture of Intel Xeon processors. As a result, the efficiency of program execution increases with increasing problem dimension. The opposite situation is observed for α_{GPU}. As the problem dimension increases, the GPU resources in implementation of the SIMD model of computations decrease, and the dependence of the execution time on the number of bodies N is enhanced. Nevertheless, the GPU operates ten times faster than the CPU, even for the maximum values of N.

CONCLUSIONS

The work performed shows that the Parker–Sochacki method can be effectively implemented on GPUs whose structure is similar to solvers with the SIMD architecture. In solving the N-body problem, the computation time is smaller than that provided by the CPU version by a factor of 10–20 for $N = 500$. For a fixed time step, an increase in the power of the Maclaurin–Taylor polynomial exerts a limited effect on the solution accuracy and fairly rapidly reaches the level of computer accuracy in accordance with the estimate of the residual term of the series. The values of the polynomial power of the order of 20 seem to be practically feasible, though it depends on the stiffness of the integrated equation.

ACKNOWLEDGMENTS

This work was supported by the Far East Branch of the Russian Academy of Sciences (project no. 09-III-A-01-04 “Parallel methods and algorithms of solving convex extreme problems of large dimensions for multithread and multiprocessor architectures”).

REFERENCES

1. Pruett, C.D., Rudmin, J.W., and Lacy, J.M., An Adaptive N-Body Algorithm of Optimal Order; http://www.sciencedirect.com/science/article/B6WHY-484SHBH-2/2/00e4db5cd90f5a49c9347cfa8b60308a.
2. Stewart, R. and Bair, W., Spiking Neural Network Simulation: Numerical Integration with the Parker–Sochacki Method, *J. Comput. Neurosci.*, 2009, vol. 27, no. 1, pp. 115–133; http://dx.doi.org/10.1007/s10827-008-0131-5.
3. Parker, G.E. and Sochacki, J.S., Implementing the Picard Iteration, *Neural, Parallel Sci. Comput.*, 1996, vol. 4, no. 1, pp. 97–112.
4. Arnold, V.I., *Obyknovennye differentsial'nye uravneniya* (Ordinary Differential Equations), Izhevsk: Izhevsk. Respubl. Tipograf., 2000.
5. Pruett, C.D., Rudmin, J.W., and Lacy, J.M., An Adaptive N-Body Algorithm of Optimal Order, *J. Comput. Phys.*, 2003, vol. 187, no. 1, pp. 298–317.
6. Belleman, R.G., Bedorf, J., and Zwart, S.P., High Performance Direct Gravitational N-Body Simulations on Graphics Processing Units; II: An Implementation in CUDA, *New Astronomy*, 2007, vol. 13, no. 2, pp. 103–112; http://dx.doi.org/10.1016/j.newast.2007.07.004.
7. Nyland, L., Harris, M., and Prins, J., Fast N-Body Simulation with CUDA, in *GPU Gems 3*, Massachusetts: Addison Wesley Professional, 2007.

8. Hamada, T., Narumi, T., Yokota, R. et al., 42 TFlops Hierarchical *N*-Body Simulations on GPUs with Applications in Both Astrophysics and Turbulence, *SC '09*: *Proc. of the Conference on High Performance Computing Networking, Storage and Analysis*, New York: ACM, 2009, pp. 1–12.
9. Fujiwara, K. and Nakasato, N., *Fast Simulations of Gravitational Many-Body Problem on RV770 GPU*, 2009; http://www.citebase.org/abstract?id=oai:arXiv.org:0904.3659.
10. Rudmin, J.W., Application of the Parker–Sochacki Method to Celestial Mechanics, Technical report of James Madison University, Harrisonburg, 1998; http://arxiv.org/abs/1007.1677v1.
11. NVIDIA. NVIDIA CUDA C Programming Guide, 3.2 ed., 2010; http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.2.pdf.
12. NVIDIA. NVIDIA CUDA Reference Manual, 2.3 ed., 2009; http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/CUDA_Reference_Manual_2.3.pdf.

View publication stats